INDUSTRIAL AI RESEARCH CENTER

# 어프렌티스 프로젝트

## 기말 프로젝트

충북대학교 산업인공지능학과 이민수

Dept. of Industrial AI

# 차례

# 1. 프로젝트 개요

INDUSTRIAL AI RESEARCH CENTER

## Cover Page

- 프로젝트명 및 팀 이름 : Meta
- 발표자 이름 : 이 민 수
- 날짜 : 2023-11-12

## 프로젝트 개요 및 데이터 소개

- 프로젝트 목표 및 문제 정의
  미국 시애틀 킹 카운티 지역의 주택 가격 영향을 미치는 요인을 찾고 가격을 예측
- 사용한 데이터셋 소개
  Kaggle에서 제공하는 House Sales in King County

# 1. 프로젝트 개요



## 프로젝트 개요 및 데이터 소개

- 데이터 출처, 크기, 형식 설명
  시애틀 킹 카운티 지역 / 20열 21,613 개의 Data
- 예측하려는 특성(종속 변수) 소개

| 변수 | 설명 | Dtype |
|---|---|---|
| price | 주택 가격(2014년 5월 ~ 2015년 5월까지의) | float64 |

# 1. 프로젝트 개요



## 프로젝트 개요 및 데이터 소개

- 특성(독립 변수)의 목록과 설명

| 변수 | 설명 | Dtype | 변수 | 설명 | Dtype |
|---|---|---|---|---|---|
| id | 집에 대한 표기 | (int64) | data | 집이 팔린 날짜 | (object) |
| bedrooms | 침실 수 | (int64) | bathrooms | 욕실 수 | (float64) |
| sqft_living | 집의 면적 | (int64) | sqft_lot | 부지의 면적 | (int64) |
| floors | 층수) | (float64) | waterfront | 물이 보이는 집 | (int64) |
| view | 조회수 | (int64) | condition | 상태 표시 | (int64) |
| grade | king county 시스템 기준으로 부여된 등급 | (int64) | sqft_above | 지하실을 제외한 집의 평방 비트 | (int64) |
| sqft_basement | 지하실의 면적(int64) | (int64) | yr_built | 건축 연도 | (int64) |
| yr_renovated | 집을 개조한 연도 | (int64) | zipcode | 우편번호 | (int64) |
| lat | 위도 좌표 | (float64) | long | 경도 좌표 | (float64) |
| sqft_living15 | 2015년 거실 면적 | (int64) | sqft_lot15 | 2015년 LotSize 면적 | (int64) |

# 2. 프로젝트 단계



- Step 1: **훈련 세트와 테스트 세트 만들기**
  - 데이터 분할 : shuffle_and_split를 사용하여 랜덤 분할
  - 분할 비율 : 훈련셋 8 : 테스트셋 2로 분할

```python
import numpy as np

def shuffle_and_split_data(data, test_ratio):
    shuffled_indices = np.random.permutation(len(data))
    test_set_size = int(len(data) * test_ratio)
    test_indices = shuffled_indices[:test_set_size]
    train_indices = shuffled_indices[test_set_size:]
    return data.iloc[train_indices], data.iloc[test_indices]
```

```python
train_set, test_set = shuffle_and_split_data(housing, 0.2)
len(train_set)
```

```
17291
```

```python
len(test_set)
```

```
4322
```

```python
from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)
```

# 2. 프로젝트 단계

INDUSTRIAL AI RESEARCH CENTER

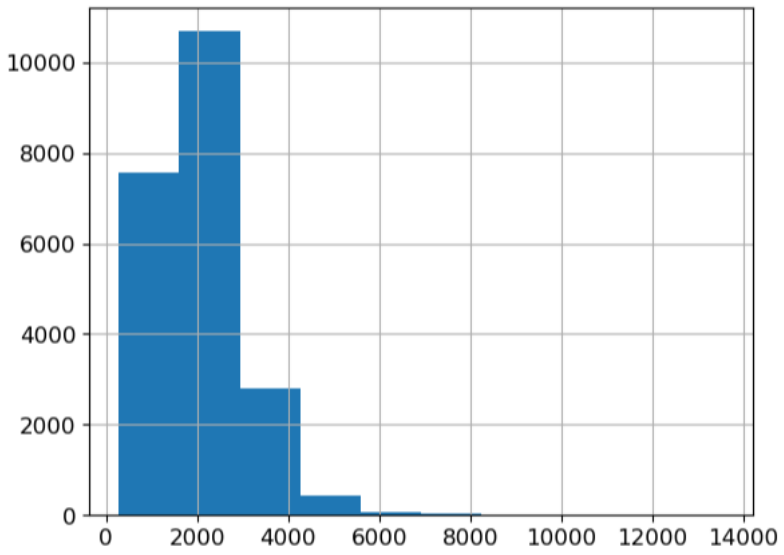
## Step 1: **훈련 세트와 테스트 세트 만들기**

- 랜덤성 및 계층적 샘플링

- 랜덤성 : np.random.seed(42) 랜덤 한 값을 생성하여 (42)로 고정하는 형태
- 계층적 샘플링 : 모집단의 데이터 분포 비율을 유지하면서 데이터를 샘플링(취즉)하는 것을 말한다. (편향)

```
housing["sqft_living"].hist()
```

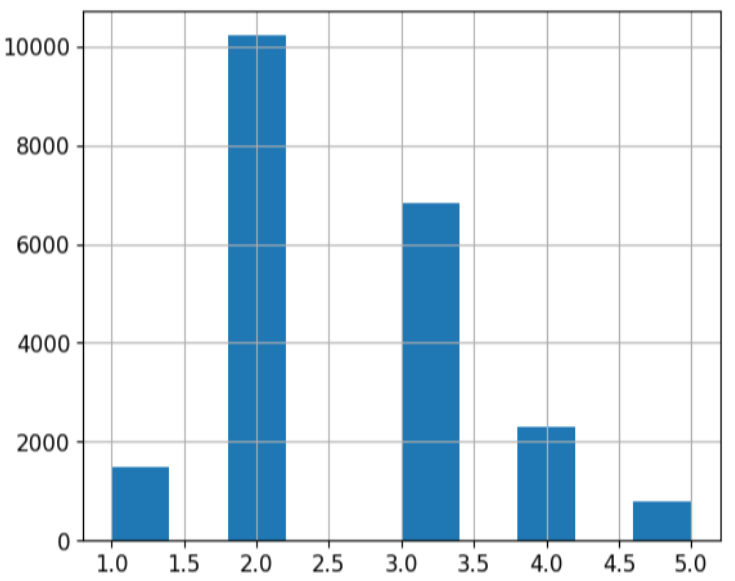
```
<Axes: >
```




```
housing["sqft_living_cat"] = pd.cut(housing["sqft_living"],
                                    bins=[0, 1000, 2000, 3000, 4000, np.inf],
                                    labels=[1, 2, 3, 4, 5])
```

```
housing["sqft_living_cat"].hist()
```

```
<Axes: >
```



# 2. 프로젝트 단계

INDUSTRIAL AI RESEARCH CENTER

## Step 2: **데이터 탐색 및 시각화**

- 데이터 피처들의 기초 통계량 (평균, 중앙값, 분산 등)

```
housing.describe()
```

| | id | date | price | bedrooms | bathrooms | sqft_living | sqft_lot | floors | waterfront | view |
|---|---|---|---|---|---|---|---|---|---|---|
| count | 2.161300e+04 | 21613 | 2.161300e+04 | 21613.000000 | 21613.000000 | 21613.000000 | 2.161300e+04 | 21613.000000 | 21613.000000 | 21613.000000 |
| mean | 4.580302e+09 | 2014-10-29 04:38:01.959931648 | 5.400881e+05 | 3.370842 | 2.114757 | 2079.899736 | 1.510697e+04 | 1.494309 | 0.007542 | 0.234303 |
| min | 1.000102e+06 | 2014-05-02 00:00:00 | 7.500000e+04 | 0.000000 | 0.000000 | 290.000000 | 5.200000e+02 | 1.000000 | 0.000000 | 0.000000 |
| 25% | 2.123049e+09 | 2014-07-22 00:00:00 | 3.219500e+05 | 3.000000 | 1.750000 | 1427.000000 | 5.040000e+03 | 1.000000 | 0.000000 | 0.000000 |
| 50% | 3.904930e+09 | 2014-10-16 00:00:00 | 4.500000e+05 | 3.000000 | 2.250000 | 1910.000000 | 7.618000e+03 | 1.500000 | 0.000000 | 0.000000 |
| 75% | 7.308900e+09 | 2015-02-17 00:00:00 | 6.450000e+05 | 4.000000 | 2.500000 | 2550.000000 | 1.068800e+04 | 2.000000 | 0.000000 | 0.000000 |
| max | 9.900000e+09 | 2015-05-27 00:00:00 | 7.700000e+06 | 33.000000 | 8.000000 | 13540.000000 | 1.651359e+06 | 3.500000 | 1.000000 | 4.000000 |
| std | 2.876566e+09 | NaN | 3.671272e+05 | 0.930062 | 0.770163 | 918.440897 | 4.142051e+04 | 0.539989 | 0.086517 | 0.766318 |

```
housing.describe()
```

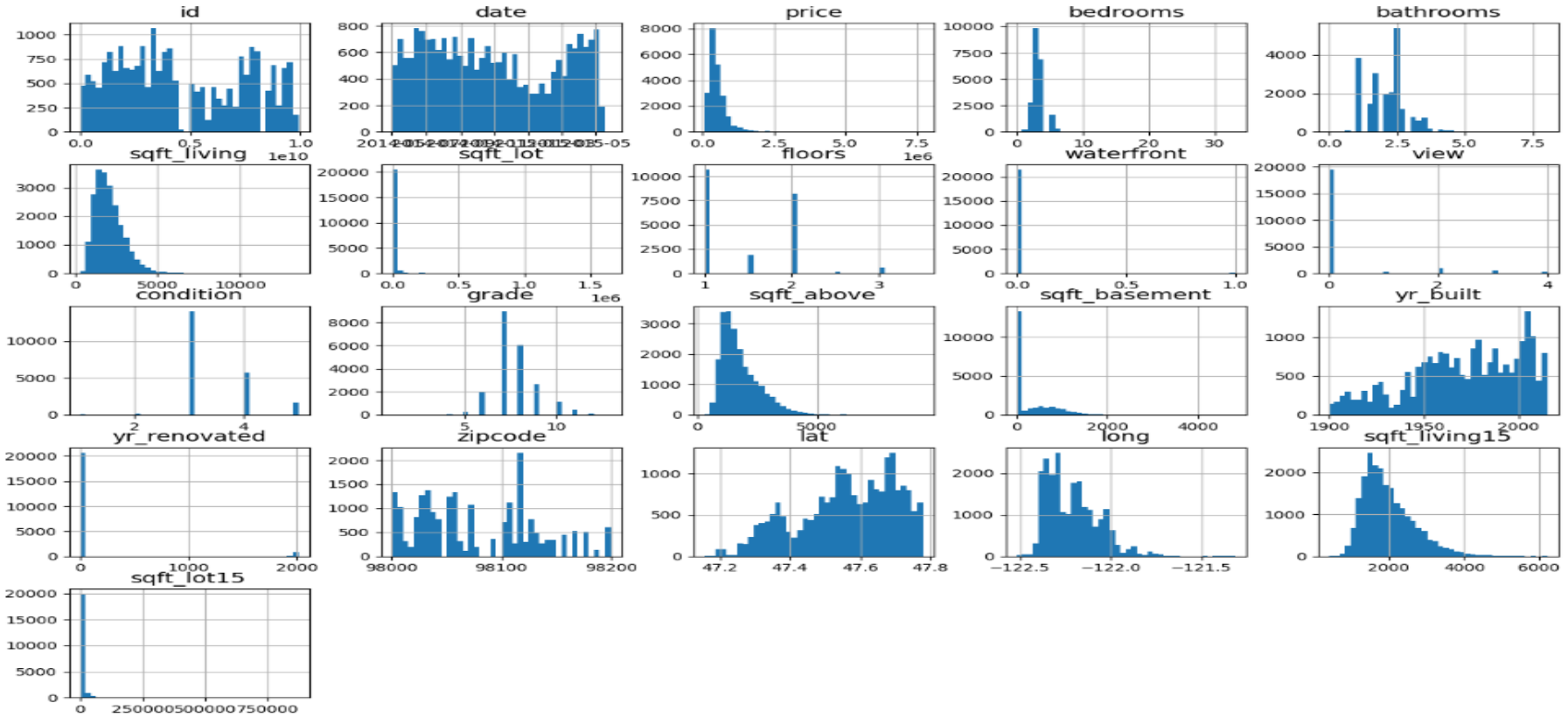
| | view | ... | grade | sqft_above | sqft_basement | yr_built | yr_renovated | zipcode | lat | long | sqft_living15 | sqft_lot15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 3.000000 | ... | 21613.000000 | 21613.000000 | 21613.000000 | 21613.000000 | 21613.000000 | 21613.000000 | 21613.000000 | 21613.000000 | 21613.000000 | 21613.000000 |
| mean | 0.234303 | ... | 7.656873 | 1788.390691 | 291.509045 | 1971.005136 | 84.402258 | 98077.939805 | 47.560053 | -122.213896 | 1986.552492 | 12768.455652 |
| min | 0.000000 | ... | 1.000000 | 290.000000 | 0.000000 | 1900.000000 | 0.000000 | 98001.000000 | 47.155900 | -122.519000 | 399.000000 | 651.000000 |
| 25% | 0.000000 | ... | 7.000000 | 1190.000000 | 0.000000 | 1951.000000 | 0.000000 | 98033.000000 | 47.471000 | -122.328000 | 1490.000000 | 5100.000000 |
| 50% | 0.000000 | ... | 7.000000 | 1560.000000 | 0.000000 | 1975.000000 | 0.000000 | 98065.000000 | 47.571800 | -122.230000 | 1840.000000 | 7620.000000 |
| 75% | 0.000000 | ... | 8.000000 | 2210.000000 | 560.000000 | 1997.000000 | 0.000000 | 98118.000000 | 47.678000 | -122.125000 | 2360.000000 | 10083.000000 |
| max | 4.000000 | ... | 13.000000 | 9410.000000 | 4820.000000 | 2015.000000 | 2015.000000 | 98199.000000 | 47.777600 | -121.315000 | 6210.000000 | 871200.000000 |
| std | 0.766318 | ... | 1.175459 | 828.090978 | 442.575043 | 29.373411 | 401.679240 | 53.505026 | 0.138564 | 0.140828 | 685.391304 | 27304.179631 |

# 2. 프로젝트 단계

INDUSTRIAL AI RESEARCH CENTER

## Step 2: **데이터 탐색 및 시각화**

- 히스토그램, 상자 그림(Box Plot) 등을 사용한 피처들의 데이터 분포 시각화



# 2. 프로젝트 단계



## Step 2: **데이터 탐색 및 시각화**

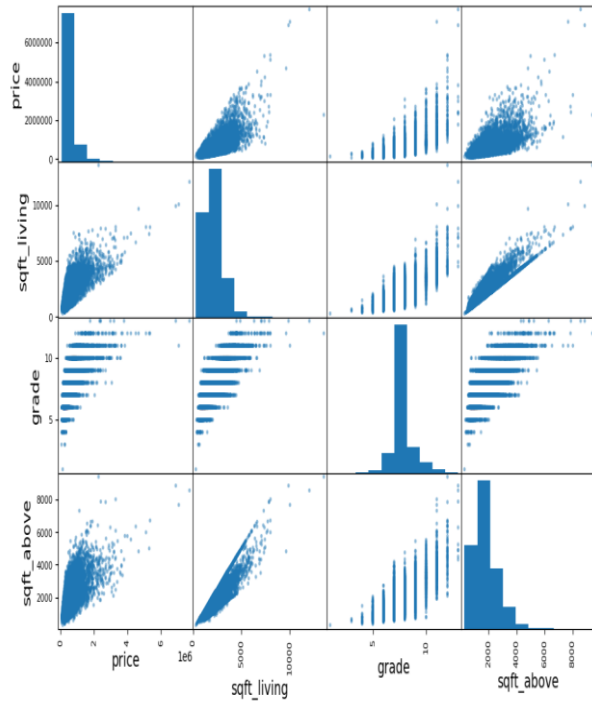
- 피처 간 상관 관계 시각화 (상관 행렬, 히트맵)

```
corr_matrix["price"].sort_values(ascending=False)
```

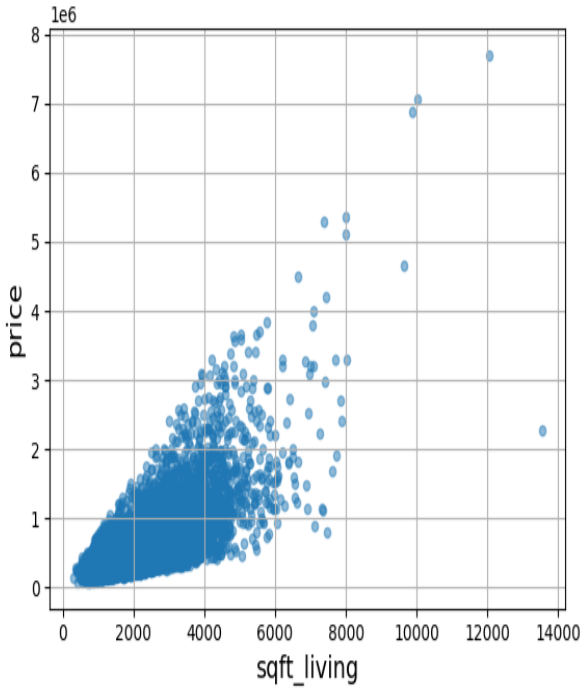
```
price          1.000000
sqft_living    0.700934
grade          0.664614
sqft_above     0.603995
sqft_living15  0.582520
bathrooms      0.524555
view           0.409998
sqft_basement  0.324851
bedrooms       0.308457
lat            0.307281
waterfront     0.285881
floors         0.261183
yr_renovated   0.129244
sqft_lot       0.091448
sqft_lot15     0.081380
yr_built       0.054315
condition      0.046525
long           0.025211
date_year     -0.000795
id            -0.017007
zipcode       -0.051180
Name: price, dtype: float64
```

```
from pandas.plotting import scatter_matrix

attributes = ["price", "sqft_living", "grade", "sqft_above"]
scatter_matrix(housing[attributes], figsize=(12, 8))
plt.show()
```




```
housing.plot(kind="scatter", x="sqft_living", y="price",
             alpha=0.5, grid=True)
plt.show()
```



# 2. 프로젝트 단계

INDUSTRIAL AI RESEARCH CENTER

## Step 2: **데이터 탐색 및 시각화**

- 피처 간 상관 관계 시각화 (상관 행렬, 히트맵)

- 특성조합 : bed_bath = bedrooms + bathrooms / living_basement = sqftr_living + sqft_basement

```
housing["bed_bath"] =  housing["bedrooms"] + housing["bathrooms"]
housing["living_basement"] = housing["sqft_living"] + housing["sqft_basement"]
```

```
price            1.000000
sqft_living      0.701768
grade            0.674241
living_basement  0.666450
sqft_above       0.605112
sqft_living15    0.589177
bathrooms        0.524402
bed_bath         0.466187
view             0.391527
sqft_basement    0.320250
bedrooms         0.308353
lat              0.307949
waterfront       0.260091
floors           0.259653
yr_renovated     0.113940
sqft_lot         0.091328
sqft_lot15       0.088731
yr_built         0.060352
condition        0.037358
```

# 2. 프로젝트 단계

INDUSTRIAL AI RESEARCH CENTER

## Step 3: **데이터 전처리**

- 범주형 데이터 처리: 더미 변수화 또는 인코딩 방법

- 범주형 특성으로 이전 단계에서 변환한 date_year 선정

- OneHotEncoder 사용(서로 독립적인 경우라고 판단함)

```
housing_cat = housing[["date_year"]]
housing_cat.head(10)
```

| | date_year |
|---|---|
| 20474 | 2014 |
| 3840 | 2014 |
| 7426 | 2014 |
| 4038 | 2015 |
| 11420 | 2014 |
| 12116 | 2014 |
| 14161 | 2014 |
| 21429 | 2014 |
| 17076 | 2015 |
| 1345 | 2014 |

```
from sklearn.preprocessing import OneHotEncoder

cat_encoder = OneHotEncoder()
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
housing_cat_1hot
```

```
<17290x2 sparse matrix of type '<class 'numpy.float64'>'
        with 17290 stored elements in Compressed Sparse Row format>
```

```
housing_cat_1hot.toarray()
```

```
array([[1., 0.],
       [1., 0.],
       [1., 0.],
       ...,
       [1., 0.],
       [1., 0.],
       [0., 1.]])
```

```
cat_encoder = OneHotEncoder(sparse=False)
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
housing_cat_1hot
```

```
array([[1., 0.],
       [1., 0.],
       [1., 0.],
       ...,
       [1., 0.],
       [1., 0.],
       [0., 1.]])
```

```
cat_encoder.categories_
```

```
[array(['2014', '2015'], dtype=object)]
```

# 2. 프로젝트 단계



- Step 3: **데이터 전처리**
  - 결측치 : SimpleImputer의 median를 사용하여 bedrooms의 13개의 값 채우기 진행

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy="median")
```

# 2. 프로젝트 단계

INDUSTRIAL AI RESEARCH CENTER

## Step 4: **데이터 스케일링**

- 사용자 정의 변환기 : 함수나 함수들의 리스트를 사용하여 데이터를 변화하고자 할 때 사용

- bed_bath 추가

```
from sklearn.base import BaseEstimator, TransformerMixin

bedrooms_ix, bathrooms_ix, sqft_living_ix, sqft_basement_ix = 3, 4, 5, 13

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_living_basement=True):
        self.add_living_basement = add_living_basement
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        bed_bath = X[:, bedrooms_ix] + X[:, bathrooms_ix]
        if self.add_living_basement:
            living_basement = X[:, sqft_living_ix] + X[:, sqft_basement_ix]
            return np.c_[X, bed_bath,
                         living_basement]
        else:
            return np.c_[X, bed_bath]

attr_adder = CombinedAttributesAdder(add_living_basement=False)
housing_extra_attribs = attr_adder.transform(housing.to_numpy())
```

```
col_names = "bedrooms", "bathrooms", "sqft_living", "sqft_basement"
bedrooms_ix, bathrooms_ix, sqft_living_ix, sqft_basement_ix = [
    housing.columns.get_loc(c) for c in col_names]
```

```
housing_extra_attribs = pd.DataFrame(
    housing_extra_attribs,
    columns=list(housing.columns)+["bed_bath"],
    index=housing.index)
housing_extra_attribs.head()
```

| sqft_lot15 | date_year | bed_bath |
|---|---|---|
| 1282 | 2014 | 1383.25 |
| 11200 | 2014 | 821.0 |
| 16440 | 2014 | 4243.5 |
| 1370 | 2015 | 1141.0 |
| 3698 | 2014 | 1602.5 |

# 2. 프로젝트 단계

INDUSTRIAL AI RESEARCH CENTER

## Step 5: **파이프라인 구성**

- 파이프라인 구조 설명

- 수치형 특성을 처리하는 파이프라인

- StandardScaler는 거리를 중요하게 생각함(특성의 스케일을 모두 일치시키도록)

- StandardScaler 데이터 열의 평균을 빼고 열의 표준편차를 나누어서 표준점수로 변환하는 전처리 방법

- SimpleImputer(meian) + CombineAttribuesAdder(특성 추가) + StandardScaler를 더하여 하나의 파이프라인으로 통합하여 이 파이프라인이 마치 하나의 변환기인 거 마냥 한 번에 변화 도록 진행

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="median")),
        ('attribs_adder', CombinedAttributesAdder()),
        ('std_scaler', StandardScaler()),
    ])

housing_num_tr = num_pipeline.fit_transform(housing_num)
```

```
housing_num_tr
```

```
array([[-1.25041026, -0.39854495,  1.47808418, ..., -0.69112335,
        -0.76486199, -0.32865802],
       [-1.41400744, -1.46639589, -1.45191173, ..., -0.69112335,
        -1.38042145, -0.10841793],
       [-0.71856641, -0.39854495,  1.80363928, ..., -0.69112335,
         2.36658123,  0.15939882],
       ...,
       [ 0.14172853, -1.46639589, -1.45191173, ..., -0.69112335,
        -0.83301419, -0.14699295],
       [-0.58110615, -0.39854495, -1.45191173, ..., -0.69112335,
        -1.0300808 , -0.25030775],
       [-0.69216179, -0.39854495,  0.17586377, ...,  1.44691972,
         1.41272408, -0.07668415]])
```

# 2. 프로젝트 단계



## Step 5: **파이프라인 구성**

- 파이프라인 적용

- 앞서 문자열을 OneHotEncoder로 변환했던 작업과 방금 위에서 합한 num 파이프라인 결합
- 20개의 열에서 24개의 열로 변경(특성추가1, date_year, 2014, 2015 = 4개)

```
from sklearn.compose import ColumnTransformer

num_attribs = list(housing_num)
cat_attribs = ["date_year"]

full_pipeline = ColumnTransformer([
        ("num", num_pipeline, num_attribs),
        ("cat", OneHotEncoder(), cat_attribs),
    ])

housing_prepared = full_pipeline.fit_transform(housing)
```

```
housing_prepared
```

```
array([[-1.25041026, -0.39854495,  1.47808418, ..., -0.32865802,
         1.        ,  0.        ],
       [-1.41400744, -1.46639589, -1.45191173, ..., -0.10841793,
         1.        ,  0.        ],
       [-0.71856641, -0.39854495,  1.80363928, ...,  0.15939882,
         1.        ,  0.        ],
       ...,
       [ 0.14172853, -1.46639589, -1.45191173, ..., -0.14699295,
         1.        ,  0.        ],
       [-0.58110615, -0.39854495, -1.45191173, ..., -0.25030775,
         1.        ,  0.        ],
       [-0.69216179, -0.39854495,  0.17586377, ..., -0.07668415,
         0.        ,  1.        ]])
```

```
housing_prepared.shape
```

```
(17290, 24)
```

# 2. 프로젝트 단계



- Step 5: **파이프라인 구성**

• 파이프라인 적용

- SimpleImputer(meian) + CombineAttribuesAdder(특성 추가) + StandardScaler를 더하여 하나의 파이프라인으로 통합하여 이 파이프라인이 마치 하나의 변환기인 거 마냥 한 번에 변화 도록 진행

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

num_pipeline = Pipeline([
        ('imputer', SimpleImputer(strategy="median")),
        ('attribs_adder', CombinedAttributesAdder()),
        ('std_scaler', StandardScaler()),
    ])

housing_num_tr = num_pipeline.fit_transform(housing_num)
```

```
housing_num_tr
```

```
array([[-1.25041026, -0.39854495,  1.47808418, ..., -0.69112335,
        -0.76486199, -0.32865802],
       [-1.41400744, -1.46639589, -1.45191173, ..., -0.69112335,
        -1.38042145, -0.10841793],
       [-0.71856641, -0.39854495,  1.80363928, ..., -0.69112335,
         2.36658123,  0.15939882],
       ...,
       [ 0.14172853, -1.46639589, -1.45191173, ..., -0.69112335,
        -0.83301419, -0.14699295],
       [-0.58110615, -0.39854495, -1.45191173, ..., -0.69112335,
        -1.0300808 , -0.25030775],
       [-0.69216179, -0.39854495,  0.17586377, ...,  1.44691972,
         1.41272408, -0.07668415]])
```

# 2. 프로젝트 단계



## Step 6: **모델 선택 및 훈련**

- 선형회귀(LinearRegression )

- LinearRgegression 객체를 기본 매게 변수로 만들고, housing_prepared 훈련셋과 housing_labels 훈련 타겟 fit 진행

- housing에 있는 원본 데이터 5개를 추출하여 실제 예측 모델 진행

```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(housing_prepared, housing_labels)
```

```
▾ LinearRegression
LinearRegression()
```

```
some_data = housing.iloc[:5]
some_labels = housing_labels.iloc[:5]
some_data_prepared = full_pipeline.transform(some_data)

print("예측:", lin_reg.predict(some_data_prepared))
```

```
예측: [ 489619.3462737   182622.486864   1158250.60464151  160618.08752826
  360509.89976217]
```

```
print("레이블:", list(some_labels))
```

```
레이블: [379000.0, 173000.0, 1390000.0, 390000.0, 440500.0]
```

# 2. 프로젝트 단계



## Step 6: **모델 선택 및 훈련**

- 평균 제곱근 오차(Root Mean Squared Error(RMSE))

- 평균 제곱근 오차의 값이 낮을수록 예측 모델이 더 좋습니다. 완벽한 예측 모델(항상 정확한 예상 값을 예측하는 가상의 예측 모델)은 평균 제곱근 오차가 값이 0 입니다.
- 약 19만 7천 달러의 차이가 나는 걸 알 수 있다.

```
from sklearn.metrics import mean_squared_error

housing_predictions = lin_reg.predict(housing_prepared)
lin_mse = mean_squared_error(housing_labels, housing_predictions)
lin_rmse = np.sqrt(lin_mse)
lin_rmse
```

```
197057.60345799619
```

# 2. 프로젝트 단계

INDUSTRIAL AI RESEARCH CENTER

- Step 6: **모델 선택 및 훈련**

• 평균 제곱 오차(Mean Absolute Error(MAE))

- 모든 절대 오차의 평균, MAE가 작을 수록 성능이 좋다고 볼 수 있다.
- price의 타겟 레이블 ?? ~ ?? 사이인데 앞서 19만 7천 달러 및 12만 4천 달러의 값이 나왔다 그리고 R-Squared의 값이 0.70 정도로 나온 것 으로 보아 좋지 않은 모델인 것으로 판단 된다
- R-Squared 값이 0.70으로 나온 것 으로 보아 충분히 훈련 데이터를 학습하지 못했기 때문에 과소 적합 된 모델인 것으로 예상 된다

```
from sklearn.metrics import mean_absolute_error

lin_mae = mean_absolute_error(housing_labels, housing_predictions)
lin_mae
```

```
124890.87415850833
```

```
from sklearn.metrics import mean_absolute_error

lin_mae = mean_absolute_error(housing_labels, housing_predictions)
lin_mae

lin_reg.score(housing_prepared, housing_labels)
```

```
0.702957648144233
```

# 2. 프로젝트 단계



## Step 6: **모델 선택 및 훈련**

- 결정트리(Decision Tree) 모델(Model)

- 전통적인 신경망을 제외하고 나머지 머신 러닝 모델 중 가장 성능이 좋은 모델이다
- R-Squared의 값이 0.99 완벽하기 때문에 훈련셋이 너무 완벽한 모델은 테스트셋 모델에서 점수가 나쁘게 나온다

```
housing_predictions = tree_reg.predict(housing_prepared)
tree_mse = mean_squared_error(housing_labels, housing_predictions)
tree_rmse = np.sqrt(tree_mse)
tree_rmse

tree_reg.score(housing_prepared, housing_labels)
```

```
0.9998978478995215
```

# 2. 프로젝트 단계



- Step 6: **모델 선택 및 훈련**

  • 교차검증(cross validation)

  - 모델의 학습 과정에서 학습/검증 데이터를 나눌 때 단순히 1번 나누는게 아니라 K번 나누고 각각의 학습 모델의 성능을 비교하여 평균 값으로 나타낸다

  - 3개의 매개변수(tree_reg, housing_prepared, housing_labels)로 10번을 나눔(cv=10)

  - 사이킷런은 cross_val_socre()의 인자로 scoring='neg_mean_squared_err'를 지정하면 반환되는 수치 값이 음수 값입니다. 사이킷런의 지표 평가 기준은 높은 지표 값일수록 좋은 모델인 데 반해, 일반적으로 회귀는 MSE값이 낮을 수록 좋은 회귀 모델입니다.

# 2. 프로젝트 단계

INDUSTRIAL AI RESEARCH CENTER

- Step 6: **모델 선택 및 훈련**

  - 교차검증(cross validation)
  - Tree모델의 값은 185,628 이며 lin모델의 값은 197,004 이다
  - 위에서 교차 검증 없이 그냥 훈련셋 점수로만 봤을 때 트레인 모델이 100% 완벽하게 보일 수 있겠지만 교차 검증을 통하여

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(tree_reg, housing_prepared, housing_labels,
                         scoring="neg_mean_squared_error", cv=10)
tree_rmse_scores = np.sqrt(-scores)
```

```
def display_scores(scores):
    print("점수:", scores)
    print("평균:", scores.mean())
    print("표준 편차:", scores.std())

display_scores(tree_rmse_scores)
```

```
점수: [172951.4350798  200098.63676544 198468.36015463 174426.06225244
 161118.22278726 215523.23026493 181172.25470477 180049.27460823
 186346.12618222 186135.96812837]
평균: 185628.95709280914
표준 편차: 14854.818838314772
```

```
lin_scores = cross_val_score(lin_reg, housing_prepared, housing_labels,
                             scoring="neg_mean_squared_error", cv=10)
lin_rmse_scores = np.sqrt(-lin_scores)
display_scores(lin_rmse_scores)
```

```
점수: [183036.30860727 206433.32513363 225665.31758434 174342.04884161
 178610.36886234 202452.79942038 182100.13611993 222667.64729352
 198478.99522196 196255.12844969]
평균: 197004.2075534668
표준 편차: 16942.67742383161
```

# 2. 프로젝트 단계



## Step 6: **모델 선택 및 훈련**

- 앙상블 학습(Ensemble Learning)
- 여러 개의 분류기(Classifier)를 생성하고 그 예측을 결합함으로써 보다 정확한 최종 예측을 도출하는 기법
- 앙상블 RandomForest
- Bagging의 대표적인 알고리즘이며 비교적 빠른 수행 속도 + 높은 예측 성능을 가지고 있다
- 앞서 진행된 평균값(197,057 124,890)에 비해 낮아진 128,233의 값을 확인함

```
from sklearn.ensemble import RandomForestRegressor

forest_reg = RandomForestRegressor(n_estimators=100, random_state=42)
forest_reg.fit(housing_prepared, housing_labels)
```

```
RandomForestRegressor(random_state=42)
```

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

```
housing_predictions = forest_reg.predict(housing_prepared)
forest_mse = mean_squared_error(housing_labels, housing_predictions)
forest_rmse = np.sqrt(forest_mse)
forest_rmse
```

```
47540.338609416365
```

```
from sklearn.model_selection import cross_val_score

forest_scores = cross_val_score(forest_reg, housing_prepared, housing_labels,
                                scoring="neg_mean_squared_error", cv=10)
forest_rmse_scores = np.sqrt(-forest_scores)
display_scores(forest_rmse_scores)
```

```
점수: [114463.49854987 125277.02693161 133579.79778414 113219.43531714
 142770.21495756 133798.0631579  116454.11088091 138430.51459657
 129260.79247501 135078.37920971]
평균: 128233.18338604113
표준 편차: 9926.126994241858
```

# 2. 프로젝트 단계



- Step 7: **모델 세부 튜닝**

• 그리드 서치(GridSearchCV)

- 머신 러닝에서 모델의 성능향상을 위해 쓰이는 기법 중 하나이며 사용자가 직접 모델의 하이퍼 파라미터의 값을 리스트로 입력하면 값에 대한 경우의 수마다 예측 성능을 측정 평가하여 비교하면서 최적의 하이퍼 파라미터 값을 찾는 과정

- RandomForestRegressor 클래스에는 n_estimators 결정 트리의 개수를 정하는 매개변수가 있습니다 위에 100으로 지정되어 있지만 여기선 3, 10, 30으로 3개를 지정하고 max_features는 일부 특성을 골라서 할 수 있는데 저는 2, 4, 6, 8개를 선택하여 테스트 진행하였습니다
결론적으로 n_estimators 3개와 max_features 4개의 조합으로 총 12개의  모델을 학습합니다

- Bootstrap 매개변수는 훈련 샘플을 랜덤하게 추출합니다 기본값이 True 인것을 False로 변경합니다 n_estimators는 3, 10으로 2개를 지정하고 max_features는 2, 3, 4로 3개를 지정하여 6개의 조합을 지도하게 하였음

# 2. 프로젝트 단계



- Step 7: **모델 세부 튜닝**

- RandomForestRegressor 클래스 객체를 만든 후 GridSearchCV에 첫 번째 매개변수로 RandomForestRegressor 클래스 객체를 넣어 주고 두번째 매개변수로 parm_grid를 설장합니다 그리고 cv=5로 지정하면서 총 90번 훈련을 진행토록 합니다

```
from sklearn.model_selection import GridSearchCV

param_grid = [
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},
  ]

forest_reg = RandomForestRegressor(random_state=42)
grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                           scoring='neg_mean_squared_error',
                           return_train_score=True)
grid_search.fit(housing_prepared, housing_labels)
```

```
GridSearchCV(cv=5, estimator=RandomForestRegressor(random_state=42),
             param_grid=[{'max_features': [2, 4, 6, 8],
                          'n_estimators': [3, 10, 30]},
                         {'bootstrap': [False], 'max_features': [2, 3, 4],
                          'n_estimators': [3, 10]}],
             return_train_score=True, scoring='neg_mean_squared_error')
```

## Step 7: **모델 세부 튜닝**

- 최고 조합 `grid_search.best_params_`

```
{'max_features': 8, 'n_estimators': 30}
```

- cv 점수 확인

```
cvres = grid_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

```
191869.30987836537 {'max_features': 2, 'n_estimators': 3}
158520.85040958205 {'max_features': 2, 'n_estimators': 10}
150703.37838298763 {'max_features': 2, 'n_estimators': 30}
170007.5766688675 {'max_features': 4, 'n_estimators': 3}
147205.75654491407 {'max_features': 4, 'n_estimators': 10}
139787.69538141083 {'max_features': 4, 'n_estimators': 30}
165847.9203830717 {'max_features': 6, 'n_estimators': 3}
142235.34908620935 {'max_features': 6, 'n_estimators': 10}
136103.92453765677 {'max_features': 6, 'n_estimators': 30}
163894.81869340595 {'max_features': 8, 'n_estimators': 3}
139096.8678715695 {'max_features': 8, 'n_estimators': 10}
132510.87292018408 {'max_features': 8, 'n_estimators': 30}
178588.8879726465 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
151197.13242860368 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
172202.40857700442 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
146554.38582948715 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
162229.24340574307 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
141060.65464689845 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```

# 2. 프로젝트 단계

INDUSTRIAL AI RESEARCH CENTER

## Step 7: **모델 세부 튜닝**

- 랜덤 서치(RandomSearchCV)

- 탐색할 파라미터의 개수를 개별로 정리해주는 것 탐색할 파라미터의 수가 너무 많거나 연속적인 숫자 값이 1인 경우

- 그리드 서치와 동일하지만 임의의 탐색할 범위를 랜덤하게 찾아서 파라미터를 탐색

- n_iter 매개변수 지정한 횟수만큼 파라미터 공학을 탐색

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

param_distribs = {
        'n_estimators': randint(low=1, high=200),
        'max_features': randint(low=1, high=8),
    }

forest_reg = RandomForestRegressor(random_state=42)
rnd_search = RandomizedSearchCV(forest_reg, param_distributions=param_distribs,
                                n_iter=10, cv=5, scoring='neg_mean_squared_error', random_state=42)
rnd_search.fit(housing_prepared, housing_labels)
```

```
RandomizedSearchCV(cv=5, estimator=RandomForestRegressor(random_state=42),
                   param_distributions={'max_features': <scipy.stats._distn_infrastructure.rv_discrete_frozen object at 0x00000207ED1CABD0
>,
                                        'n_estimators': <scipy.stats._distn_infrastructure.rv_discrete_frozen object at 0x00000207ED1CB450
>},
                   random_state=42, scoring='neg_mean_squared_error')
```

**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

# 2. 프로젝트 단계



- Step 7: **모델 세부 튜닝**
  - 최고 결과

```
cvres = rnd_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

```
130872.34752893592 {'max_features': 7, 'n_estimators': 180}
142637.42624116506 {'max_features': 5, 'n_estimators': 15}
139955.3631568677 {'max_features': 3, 'n_estimators': 72}
140056.8826867332 {'max_features': 5, 'n_estimators': 21}
131328.38446096145 {'max_features': 7, 'n_estimators': 122}
139992.4951947791 {'max_features': 3, 'n_estimators': 75}
139708.36258192183 {'max_features': 3, 'n_estimators': 88}
134614.8619527555 {'max_features': 5, 'n_estimators': 100}
138519.75821530246 {'max_features': 3, 'n_estimators': 150}
187637.00331330727 {'max_features': 5, 'n_estimators': 2}
```

# 2. 프로젝트 단계



## Step 8: **결과와 결론**

- Feature_importances : 가장 좋은 모델을 학습한 객체의 특성을 확인하기 위함

- sqft_living의 특성이 가장 좋은 특성으로 메뉴에 점수가 0.17 로 보여진다
- 특성조합한 bed_bath가 상위 4번째에 있는 것을 확인할 수 있다

```
extra_attribs = ["bed_bath", "living_basement"]
cat_encoder = full_pipeline.named_transformers_["cat"]
cat_one_hot_attribs = list(cat_encoder.categories_[0])
attributes = num_attribs + extra_attribs + cat_one_hot_attribs
sorted(zip(feature_importances, attributes), reverse=True)
```

```
[(0.17476431576252244, 'sqft_living'),
 (0.1687772805610114, 'grade'),
 (0.14177307146991416, 'lat'),
 (0.13262940954961672, 'bed_bath'),
 (0.08522478400664532, 'sqft_living15'),
 (0.057229111770469646, 'long'),
 (0.037066022317774366, 'sqft_above'),
 (0.030972890222178327, 'yr_built'),
 (0.026868177681663115, 'bathrooms'),
 (0.026671913672797747, 'view'),
 (0.026070237277962798, 'zipcode'),
 (0.016271817785396128, 'sqft_lot15'),
 (0.01534125873280499, 'waterfront'),
 (0.011361675742605116, 'sqft_lot'),
 (0.011313730212614734, 'living_basement'),
 (0.01002902429982905, 'id'),
 (0.008775994223954222, 'sqft_basement'),
 (0.00489404975923619 5, 'bedrooms'),
 (0.004857172072296021, 'condition'),
 (0.002732265689122553, 'floors'),
 (0.0024167338510514033, 'yr_renovated'),
 (0.001374765422095587, '2014'),
 (0.001331058854254098, 'date_year'),
 (0.0012532390621838763, '2015')]
```

# 2. 프로젝트 단계



## Step 8: **결과와 결론**

- 테스트셋으로 시스템 평가

- x_test price 특성 drop, y_test price에 복사하여 준비
- full_pipeline.tranform을 X_test 적용 후, final_predictions를 만든다
- y 값이 얼마나 차이나는지 rmse로 값을 도출 한다

```
final_model = grid_search.best_estimator_

X_test = strat_test_set.drop("price", axis=1)
y_test = strat_test_set["price"].copy()

X_test_prepared = full_pipeline.transform(X_test)
final_predictions = final_model.predict(X_test_prepared)

final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)
```

```
final_rmse
```

```
145798.93323909858
```

- Step 8: **결과와 결론**

  - 테스트 RMSE에 대한 95% 신뢰 구간은 아래와 같다

```
from scipy import stats

confidence = 0.95
squared_errors = (final_predictions - y_test) ** 2
np.sqrt(stats.t.interval(confidence, len(squared_errors) - 1,
                         loc=squared_errors.mean(),
                         scale=stats.sem(squared_errors)))
```

```
array([124056.99968971, 164695.22972845])
```

- Step 8: **결과와 결론**

  - 테스트 RMSE에 대한 95% 신뢰 구간은 아래와 같다

```
from scipy import stats

confidence = 0.95
squared_errors = (final_predictions - y_test) ** 2
np.sqrt(stats.t.interval(confidence, len(squared_errors) - 1,
                         loc=squared_errors.mean(),
                         scale=stats.sem(squared_errors)))
```

```
array([124056.99968971, 164695.22972845])
```

# Thank You!